

## Extending PSSA for Complex Systems

Professor John McDermid, Department of Computer Science, University of York, UK  
Dr Mark Nicholson, Department of Computer Science, University of York, UK

Keywords: preliminary system safety assessment, human factors

### Abstract

Preliminary System Safety Assessment (PSSA) is a key stage of the safety process in the civil aerospace community. It is identified in ARP 4754/4761 as the stage in the safety process concerned with validating systems architecture, and producing derived safety requirements on system components. A very similar approach has been adopted by EUROCONTROL for Air Traffic Management (ATM).

The process in ARP 4754, and the associated guidance in ARP 4761 (on which the EUROCONTROL work is based) focuses on technical systems. However, modern aircraft systems and ATM are complex and involve computers, hence software, human operators (pilots or Air Traffic Controllers (ATCOs)) and procedures. Thus, to be effective, PSSA has to deal with people, procedures and software, as well as the more classical technical issues covered by current standards. Also, the analysis of human behaviour needs to deal both with success, e.g. correct mitigation of equipment failure, and failure, e.g. lapses in implementing procedures.

The paper presents an approach to extending PSSA to deal with software, human and procedural issues being developed in collaboration with EUROCONTROL, and the current (European and US) activities to update ARP 4754.

### Introduction

The civil aerospace guidelines ARP 4754 (ref. 1) and 4761 (ref. 2) set out requirements for the system safety process as applied to aircraft (and engines). ARP 4754 is the “higher level” document dealing with general certification. ARP 4761 gives a more detailed definition of the safety process, and presents a worked example of the process in Appendix L. The guidelines identify several phases of the system safety process, see figure 1. The key role of Preliminary System Safety Assessment (PSSA) is to act as a design driver, specifically to ensure that the design takes into account safety requirements derived from early hazard and safety analyses.

At the time when the ARPs were first published we produced an analysis of difficulties which we perceived with the standards (ref. 3), especially as their stated role is to deal with *complex and integrated systems*. Many of these issues are still valid, have been accepted, and are being addressed in an activity undertaken by a EUROCAE WG63 working group. We do not repeat discussion of those difficulties here. Instead the focus is on software and human factors issues which have a major contribution to make to the safety of complex integrated (control) systems.

In reference 3 we focused mainly on difficulties with PSSA. Our aim here is to illustrate how we believe that software and human factors issues can be addressed in the PSSA process. These suggestions arise out of work with the aircraft industry and in preparing and presenting training material for EUROCONTROL, who are responsible for European Air Traffic Management.

The paper starts with a brief overview of the ARP 4754/4761 process, then outlines the proposed extensions to the PSSA process, using a simple (non-aerospace) example for illustrative purposes.

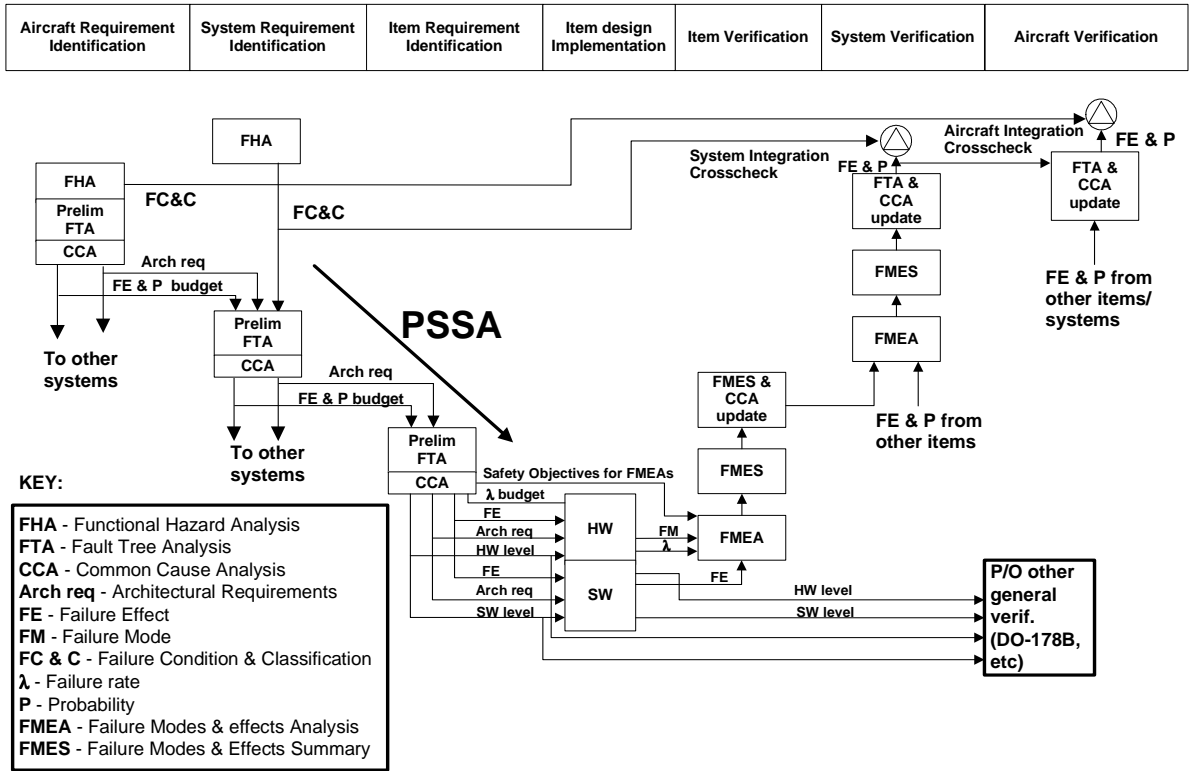


Figure 1: The ARP safety assessment process

### ARP4761 Process

In the ARP4761 process, hazard assessment involves functional hazard assessment (FHA) at the aircraft and system levels, often using functional failure analysis (FFA). PSSA is concerned with analysing proposed system designs (architectures) to validate the safety of the design, and to identify derived safety requirements (DSRs) which will guide further development of the design. Typical DSRs include budgeted failure rate requirements for components of the system architecture and development assurance levels (DALs). Although the ARPs do discuss DALs there is a clear focus on technical systems, and minimal treatment of software and human factors.

ARP 4754 identifies the role of PSSA as follows: “PSSA is a systematic examination of the *proposed system architecture(s)* to determine how failures can cause the functional hazards identified by the FHA. The objective of the PSSA is to *establish the safety requirements* of the system and to determine that *the proposed architecture can reasonably be expected to meet the safety requirements* identified by the FHA.” (ref. 1, our *emphasis*). When software and humans – pilots or air traffic controllers (ATCOs) – are part of the system the issue is how to extend the approach to determine appropriate safety requirements, and how to validate the architecture with software, human and equipment elements.

There are two key elements here:

- Producing DSRs – establishing requirements on the system components which, if they are met, will enable the architecture to meet its safety requirements. Here the components must include the software and humans. In the software case, DSRs will address failure of primary functions and provision of safety functions, e.g. to mitigate failures. In the case of humans, the DSRs will include detection of hazardous situations, and of procedures to mitigate them.

- Validating the architecture – showing that the architecture is *credible* as a way of meeting the safety requirements derived in the FHA phase. In validating the architecture we are seeking to reduce the risk of reaching the end of the development process and finding that the system is not certifiable – hence introducing cost and time over-runs into the process. This means that we have to have confidence that humans and software can meet the associated DSRs.

An issue in producing DSRs and validating architectures for complex integrated systems is to determine what are credible failure modes for software and humans and, to what extent, it is possible to estimate the rates of occurrence of such failures.

### PSSA by Example

We present our ideas by considering a very simple chemical process plant design, see Figure 2. The design has two major sub-systems. The equipment under control comprises the tank, the manually operated valve A which controls the flow out of the tank, and associated piping. There is also an automated protection sub-system comprising the two level sensors (X and Y), the controller and the automatically actuated valve B (the Xs in the figure show that loss of the sensors leads to a loss of the protection function). The key hazard associated with the system is overflow of the tank. Due to the nature of the operation in the vicinity of the tank this is viewed as being only major, and a target of  $10^{-5}$  per hour is set for the hazard, for illustrative purposes.

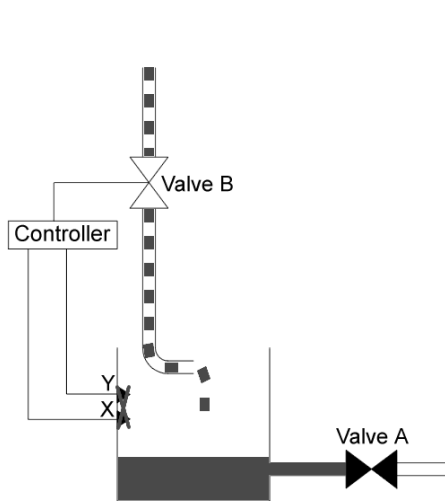


Figure 2: Tank Example

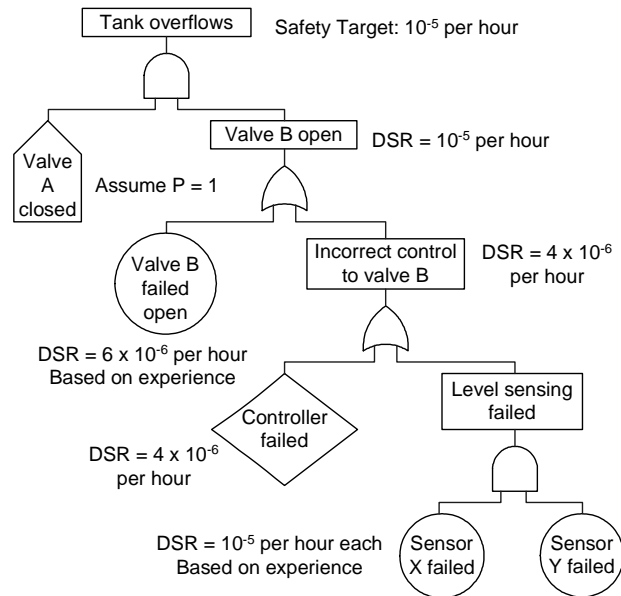


Figure 3: Tank PSSA Fault Tree

The fault tree analysis for this hazard is summarised in Figure 3, where failure rate targets are propagated down the fault tree, as DSRs, with respect to the identified failure modes. The allocation is based on experience and shows that the design is unsatisfactory in several regards, two of which relate to the controller. First, the controller is a single point of failure which can give rise to a hazard. Second, the failure rate of a simplex controller is estimated at  $10^{-4}$  per hour, based on historical data, whereas the target is  $4 \times 10^{-6}$  per hour. Thus there is a qualitative DSR for redundancy. Note: the failure rates in Figure 3 are targets or requirements, not achievements and need to be verified in design and implementation, via System Safety Analysis (SSA).

## Extensions to PSSA for Software

The aim of PSSA is to maximise the flow of safety related information to system designers. Thus the failure event “computer fails” in the fault tree shown in Figure 3 is of very little value since the corresponding DSR would be “controller doesn’t fail”! This is unrealistic, can’t be verified, and offers no guidance to the designer. Something better is needed. This issue is not properly addressed by the current standard.

We begin by viewing the controller as a “black box” and identify system level functional failures which are meaningful to designers, such as “failure to close valve B when liquid level high” (as indicated by sensors X and Y) and “do not open valve B once liquid level has reached ...”.

We next need to understand how the software can contribute to these functional failures. This can be done by applying FFA principles. For computer system FFA, we use a simple failure model with three elements: function not provided when intended, for example “not closing valve B ...”, function provided when not intended, for example “reopening valve B ...” and function provided incorrectly, for example partial closure or late closure of valve B. Conditions which could cause these failures, e.g. stale sensor data, will be considered at the design level.

In Figure 4 the fault tree is extended to show “functional” software failure modes. Note that software failure modes are (initially) hypothetical and at some point in the design process the hypothetical software failure modes put forward here must either be shown to be impossible or be shown to be sufficiently unlikely to occur so that the risk associated with them is acceptable.

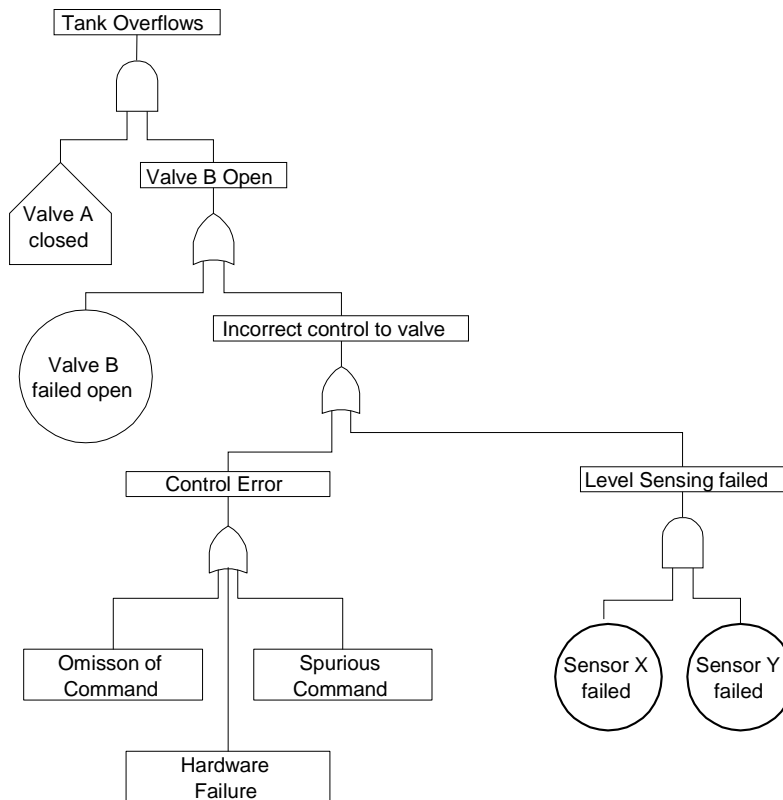


Figure 4: Extended Tank PSSA FTA

This analysis can now be used to help determine the validity of the design postulated for this system. That is, will software give rise to a hazardous failure mode (violate one of the DSRs) under credible conditions? This requires a mapping from software behaviour to real-world effects for instance, for the tank controller in this design, one hazardous state is (X = High or Y = High) and Valve B not commanded to shut. The aim of the analysis should be to show that this cannot occur or is acceptably unlikely to occur. How can this be shown?

Validation requires inspection or analysis of the software specification. The easiest way to specify the software in this case is via a simple state machine, see Figure 5. In Figure 5 the software (and controller) starts (is initialised) at the black blob, then moves into the state (shown as a circle) known as Level Normal, corresponding to the water being below the level sensors. The state machine makes the transition from Level Normal to Level High if either of the conditions X = High or Y = High are true, i.e. if either sensor detects the water. The conditions are shown in square brackets “[ ... ]”, and an action appears after the “/”, i.e. Close B (valve B), in this case. So does the current tank specification violate the DSR? Is there a path which omits the command? In fact given the model in Figure 5 this cannot occur for the “first alarm” but could on the second as it does not return to Level Normal! Also, a problem could arise on initialisation – as the specification assumes that the water level is low when the system starts. By way of illustration, a DSR could be produced that focuses on the first alarm, such as – not ((X = High or Y = High) and State = Level Normal).

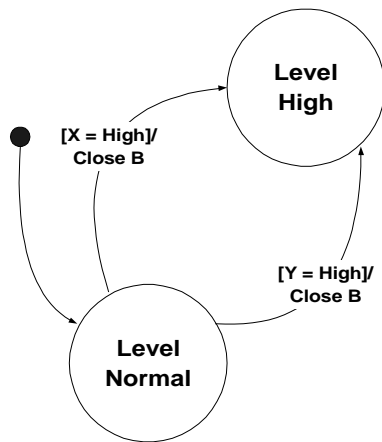


Figure 5: Black Box Software Specification

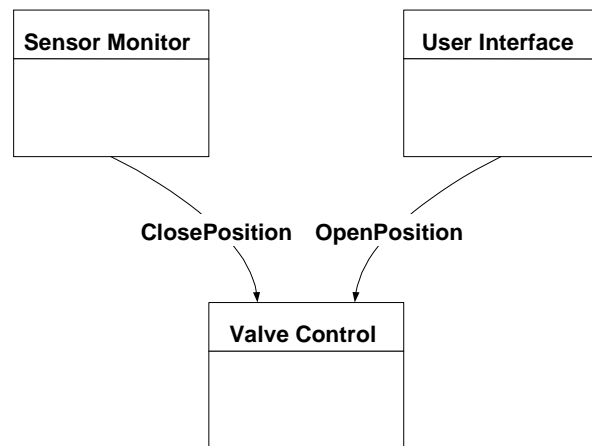


Figure 6: Software Architecture

Problems detected at this stage result in a specification change. There is clear evidence that the majority of problems in safety-critical software-based control systems are with specifications/requirements. Often, a software specification is very idealised and ignores initialisation, failure conditions, etc. In this case large parts of the functionality are left to software engineers to determine. The PSSA process is designed to guide the safety aspects of the design so the design cannot be validated unless it can be shown that the software initialises the system into a safe state and that the software preserves system safety through failures.

We can now consider a simple software architecture, as shown in figure 6. Here we have modules to monitor the sensors and control valve B – which are clearly necessary from the specification – and a further module which allows an operator to reopen valve B once the water level has dropped below the level sensors. It would be possible to analyse the software using a form of

FFA, but we have found it more practical to work in terms of a variant of HAZOP (ref. 4), known as SHARD (ref. 5) which analyzes flows. Here we consider the effect of deviations in flows between elements of the architecture (software modules) and establish DSRs for the components at each end of the flow. This is illustrated in Table 1 below, where two letter abbreviations are used for module names, i.e. Sensor Monitor is SM, Valve Control is VC and User Interface is UI.

Table 1: Partial HAZOP on Software Architecture

Data Flow	Guide Word	Deviation	Possible Causes	Consequences	Action
SM to VC	Omission	Close valve command not sent when level high	<ul style="list-style-type: none"> <li>• Sensor failures</li> <li>• Design error</li> <li>• Object not scheduled</li> </ul>	Fluid will overflow. Potentially hazardous.	Analyse design and consider how to verify code and scheduling.
SM to VC	Late	Close valve command sent tardily	<ul style="list-style-type: none"> <li>• Scheduling error</li> <li>• Excessive worst case execution time</li> </ul>	Fluid will rise towards top of container. May be hazardous.	Identify response time required
UI to VC	Commission	Valve opened while A still closed (and level high)	<ul style="list-style-type: none"> <li>• Operator error</li> <li>• Design error</li> <li>• Mis-scheduling of maintenance code</li> </ul>	Fluid will overflow, unless operator notices and corrects. Potentially hazardous.	Add interlock to prevent operator from inadvertent operation. Consider how to verify code and scheduling.

The analysis shows that there are problems with the design. For example the analysis of the UI to VC flow shows that the Operator can open Valve B when the water level is high – this is a direct cause of the Spurious Command event in Figure 4, and is hazardous. Thus there is a DSR for an interlock to prevent this – and this requires a further flow from the Sensor Monitor to the User Interface, i.e. a design revision as well as a functional DSR on the User Interface module. The fault tree shown in Figure 4 could be extended (below Spurious Command) to show the associated failure event, i.e. inadvertent opening of Valve A. The fault tree would show the operator error as a base event, and place a DSR (including a probability of failure on demand) on the software to detect and mitigate the operator error, using the interlock. Thus the DSR provides a key part of the link between the safety process and the design process.

Note also that there are other types of DSR. That relating to an Omission of the SM to VC flow is a *process* DSR for verification that the software behaves as specified. That relating to “Late” arrival of the SM to VC flow is again a process requirement, but this time to clarify requirements. In general, DSRs will include process issues, as well as product issues.

A key aim of PSSA is to support validation of the proposed architecture. In part this requires determining whether or not the quantitative elements of the DSRs are credible. In general software can give rise to (hazardous) failures if it has erroneous or missing inputs, or there is an internal failure – i.e. a “bug” is triggered. There is some controversy about the use of rates of occurrence of software failures – but we believe there is not an issue when it comes to design validation:

- Rate of occurrence  $10^{-2}$  to  $10^{-4}$  per hour or per demand – within the range of statistical testing, credible for high quality software processes and for mature commercial products;
- Rate of occurrence  $10^{-4}$  to  $10^{-6}$  per hour or per demand – beyond the limits of statistical testing. Requires a specialist high integrity software process. Not realistic for commercial products.
- Rate of occurrence  $< 10^{-6}$  per hour or per demand – beyond what can normally be attained, and indicates high risk that requirement will not be met.

Verifying that the DSRs have been met is an issue for the SSA process. Space does not permit a full analysis, although we have previously questioned the credibility of claims based simply on DALs (ref. 7). At minimum, evidence should be sought from the company developing the software that the DSRs are achievable, rather than relying on the above “rules of thumb”.

### Extensions for Human Factors and Procedures

Implicitly the analysis of the software above assumed the possibility of human error. If there was no chance that the operator would command Valve B to open when the water level is high then there is no need for the interlock. It is well-known, however, that humans are fallible – the issue is what is the credible failure behaviour, and how do we establish appropriate DSRs?

To answer these questions we need to include a model of interaction between the operator and the system (see Figure 6) and the types of cognitive error which might occur at each stage of the interaction (see table 2). This then gives a basis for analysis, again in HAZOP-like style. Figure 6 and Table 2 show two of the many classifications of interaction and cognitive failure, drawn from the THEA method (ref. 6); our aim here is to illustrate the approach, not to argue the merits or demerits of any particular method.

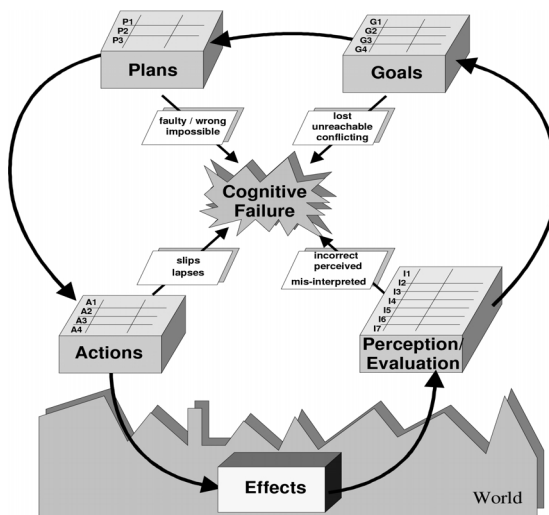


Figure 6: Simple Model of Interaction

Table 2: Analysis Guidewords

Stage	Guidewords
Perception/ Evaluation	Failure to perceive correctly Misinterpretation
Goals	Lost, unachievable, conflicting No triggering/activation Triggering activation at wrong time Wrong goal activated
Plans	Faulty Wrong Impossible
Actions	Slips Lapses

Using the above model and guidewords it is possible to model human failure – but what is the “design” which is to be analysed? As with software, it is possible to do a “high level” analysis, just in terms of the definitions of goals, etc. This is analogous to FFA. A lower level analysis can then be carried out on procedures – and this is analogous to HAZOP.

Table 3: Illustration of Analysis of Human Goals and Tasks

Stage	Items	Deviation	Interpretation	Action
Perception/ Evaluation	1 Assess fluid level	Failure to perceive correctly	Think falling when not	Signal state of A to operator via interface
	2 Assess state of Valve A		Think open when not	
Goals	1 Maximise flow 2 Do not trigger overflow	Conflict	Goals 1 and 2 conflict	Training?
		No trigger	Goal 2 no trigger	Interlock – see above
		Trigger at wrong time	Goal 1 triggered when level high	
Plans	Open Valve B when A open and level falling	Faulty	No direct evidence of state of Valve A	Signal state of A to operator via interface
Actions	Open Valve B	Slip	Improbable – sole action	-

Table 3 gives a partial definition of the goals and tasks to be undertaken by the operator for the tank example, and presents the results of the analysis, using the guidewords in Table 2. The analysis of the goals identifies the need for the interlock already determined by the analysis of the software. In addition there are DSRs on the interface, specifically to make information available to the operator. Thus the analysis drives out DSRs on the interface – but there will also be DSRs on the operator not to initiate undesirable event sequences and to mitigate unintended events.

The approach illustrated here can be extended to the analysis of procedures, applying the deviations above to the steps in the procedures, according to whether the step is concerned with perception, goal formation, and so on. Space does not permit the inclusion of an example, but it is worth noting that the analysis can produce DSRs on training and modifications to procedures, as well as DSRs on the system and software.

As with software, allowable rates of occurrence of these “deviations” can be allocated from extended PSSA fault trees. Again there is the issue of validation of the DSRs, i.e. are the rates allocated to pilots or ATCOs credible? Similarly, it is controversial to put figures on rates of occurrence of human error. However, for validation there are some “rules of thumb” (based on data from many sources):

- $5 \times 10^{-5}$  to  $5 \times 10^{-3}$  per demand for “automatic” acts, such as changing a gear in a car;
- $5 \times 10^{-4}$  to  $5 \times 10^{-2}$  per demand for “rule-based” acts, such as navigating a junction;
- $5 \times 10^{-3}$  to  $5 \times 10^{-1}$  per demand for “knowledge-based” acts, such as planning a route.

These figures help in validating designs, but more detailed analysis will normally be useful. For example, consideration of the other tasks being undertaken at the same time as the action under consideration. In other words, the cognitive workload at the time of action must be considered to provide a complete validation. Again space does not permit a full analysis of the possibilities.

A final point to note is the need to deal with common cause events. In many cases, humans can be a common cause of an accident, e.g. if they fail in one act of perception, they may well fail in a second. Thus allocation and validation of DSRs needs to be undertaken in the context of the procedure and fault tree representing the operational scenario. The same also applies to software, but space does not permit an analysis of these common cause issues.

## EUROCONTROL Collaboration

EUROCONTROL are developing guidelines for hazard and safety analysis of Air Traffic Management (ATM) systems, relying on PSSA to “drive the ATM system design” from the safety perspective. ATM systems are very complex, and involve large software systems – often running into millions of lines of code – supporting large numbers of operators, including ATCOs. The software in these systems is often not safety critical, but a combination of human error and software malfunction can be hazardous, e.g. leading to violations of minimum separation. Thus it is important to analyse the software and human behaviour of ATM systems in context, particularly taking into account their interactions.

EUROCONTROL are developing and refining their procedures for ATM hazard and safety analysis, and we are assisting in preparing and presenting training material for EUROCONTROL staff. The work described above has been developed partially in response to this training need.

### Extending ARP 4754 (ED-79) / 4761

The ARPs have been in use for about seven years, and considerable experience has been gained with their use. This has led to a number of comments on the standards, and it was agreed in 2002 that the standards needed to be revised. WG63 has been set up to develop material to support development and certification of aircraft systems and to co-ordinate guidance for complex systems, their safety / reliability, their software and hardware items and their modular implementations. EUROCAE is co-ordinating with the S-18 committee to update the ARP4754 / ED-79 document. The currently proposed date for a draft is the summer of 2005. The material discussed in this paper explains part of the University of York’s submission to this WG.

### Conclusions and Future work

ARP 4754 and ARP 4761 are intended to set in place system a safety engineering processes for complex integrated systems. Many such systems contain substantial amounts of software, and it is common for operators, e.g. pilots and ATCOs, to have an important role in achieving safety. Despite this, the current standards do not address software and human factors fully.

We have outlined an extension to the PSSA process which addresses software and human factors more directly. The process builds on existing techniques and integrates them into the overall PSSA process. The process has been illustrated with a simple example, and thus may seem rather simplistic. However an important facet of what we propose is that it is a simple and natural extension of “classical” PSSA, and should be accessible to system safety engineers who are not experts in software or human factors.

As set out, the process enables analysts to make a critical assessment of software specifications and procedures, and to investigate the interaction of software and users of a computer-based system. This enables derived safety requirements to be established on the software, procedures and training – as well as other important issues, e.g. software verification. There are many assumptions about the nature and quality of the specifications. Space does not permit a detailed analysis of the issues but, as with any safety analysis, its validity and utility is strongly dependent on the quality of the underlying design models.

In this brief paper we have not been able to address all the key issues for PSSA (and certainly not for the ARPs as a whole). For example, we have not addressed integrated modular avionics

(IMA) although a companion paper (ref. 8) gives an analysis process for IMA which is intended to be compatible with the PSSA process defined here. There are issues of scalability which are outside the scope of this paper, but we note that many of the techniques outlined here offer the possibility of (partial) automation.

Our work with EUROCONTROL and our involvement in the evolution of ARP 4754 and ARP 4761 gives us confidence in the principles presented, and the ability to get them adopted in practice. An issue for the revision to the ARPs is to create a requirement for such analysis to be carried out, rather than mandating the use of particular methods. In this way we hope to add value to these activities, without creating unnecessary constraints.

### References

1. Society of Automotive Engineers Inc, Aerospace Recommended Practice (ARP) 4754: Certification Considerations for Highly-Integrated or Complex Aircraft Systems, November 1996.
2. Society of Automotive Engineers Inc, Aerospace Recommended Practice (ARP) 4761: Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment, December 1996.
3. S Dawkins, J A McDermid, J Murdoch, D J Pumfrey, Issues in the Conduct of PSSA, Proceedings of ISSC 1999, Systems Safety Society, 1999.
4. T Kletz, HAZOP and HAZAN, Butterworth Heinemann, 1999.
5. Pumfrey, D.J., The Principled Design of Computer System Safety Analyses, DPhil, Department of Computer Science, University of York, 2000.
6. S Pocock, M Harrison, P Wright, P. Johnson, THEA: A technique for human error assessment early in design, In Hirose, M., (Ed.), Human-Computer Interaction INTERACT'01 IFIP TC.13 International Conference on human computer interaction, pages 247-254. IOS Press, 2001
7. J A McDermid, Software Safety: Where's the Evidence?, Australian Workshop on Industrial Experience with Safety Critical Systems and Software, P A Lindsay (Ed.), Australian Computer Society, 2001.
8. P M Conmy, J A McDermid, M Nicholson, Identifying Safety Dependencies in Modular Computer Systems, ISSC 2003.

### Biography

Prof. John McDermid has been Professor of Software Engineering at the University of York since 1987 where he runs the High Integrity Systems Engineering (HISE) research group. HISE studies a broad range of issues in systems, software and safety engineering, and works closely with the UK aerospace industry. Professor McDermid is the Director of the Rolls-Royce funded University Technology Centre (UTC) in Systems and Software Engineering and the BAE SYSTEMS-funded Dependable Computing System Centre (DCSC). He is author or editor of six books, and has published about 280 papers.

Dr. Mark Nicholson is a teaching and research fellow in the HISE group at York. He attained his PhD in 1998 examining process allocation in real-time distributed systems. He has over seven years of experience examining safety assessment of real-time systems. His current research interests include IMA certification processes and the evidence required for use of off-the-shelf operating systems in safety critical applications. He is a member of the MATISSE project (grant GR/R70590/01) and EUROCAE WG63 working group that is updating ED-79 (ARP4754).